Thesis for The Degree of Doctor of Philosophy

Approximation and Compression Techniques to Enhance Performance of Graphics Processing Units

Alexandra Angerd

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology
Göteborg, Sweden, 2020

Approximation and Compression Techniques to Enhance Performance of Graphics Processing Units
Alexandra Angerd

Göteborg, Sweden, 2020

ISBN 978-91-7905-425-0

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 4892
ISSN 0346-718X

Technical Report No. 192D
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Printed by Chalmers Digitaltryck
Chalmers Tekniska Högskola
Göteborg, Sweden 2020

Approximation and Compression Techniques to Enhance Performance of Graphics Processing Units

Alexandra Angerd
Department of Computer Science and Engineering
Chalmers University of Technology

Thesis for the Degree of Doctor of Philosophy

Abstract

A key challenge in modern computing systems is to access data fast enough to fully utilize the computing elements in the chip. In Graphics Processing Units (GPUs), the performance is often constrained by register file size, memory bandwidth, and the capacity of the main memory. One important technique towards alleviating this challenge is data compression. By reducing the amount of data that needs to be communicated or stored, memory resources crucial for performance can be efficiently utilized.

This thesis provides a set of approximation and compression techniques for GPUs, with the goal of efficiently utilizing the computational fabric, and thereby increase performance. The thesis shows that these techniques can substantially lower the amount of information the system has to process, and are thus important tools in the process of meeting challenges in memory utilization.

This thesis makes contributions within three areas: controlled floating-point precision reduction, lossless and lossy memory compression, and distributed training of neural networks. In the first area, the thesis shows that through automated and controlled floating-point approximation, the register file can be more efficiently utilized. This is achieved through a framework which establishes a cross-layer connection between the application and the microarchitecture layer, and a novel register file organization capable of leveraging low-precision floating-point values and narrow integers for increased capacity and performance.

Within the area of compression, this thesis aims at increasing the effective bandwidth of GPUs by presenting a lossless and lossy memory compression algorithm to reduce the amount of transferred data. In contrast to state-of-the-art compression techniques such as Base-Delta-Immediate and Bitplane Compression, which uses intra-block bases for compression, the proposed algorithm leverages multiple global base values to reach a higher compression ratio. The algorithm includes an optional approximation step for floating-point values which offers higher compression ratio at a given, low, error rate.

Finally, within the area of distributed training of neural networks, this thesis proposes a subgraph approximation scheme for graph data which mitigates accuracy loss in a distributed setting. The scheme allows neural network models that use graphs as inputs to converge at single-machine accuracy, while minimizing synchronization overhead between the machines.

Keywords: Floating-Point Precision, GPU, Approximate Computing, Microarchitecture, Compression, Register File, Machine Learning

List of Publications

This thesis is based on the work contained in the following papers:

I. **Alexandra Angerd**, Erik Sintorn, and Per Stenström. "A Framework for Automated and Controlled Floating-Point Accuracy Reduction in Graphics Applications on GPUs", *ACM Transactions on Architecture and Code Optimization (TACO), Volume 14 Issue 4, December 2017* .

II. **Alexandra Angerd**, Erik Sintorn, and Per Stenström. "A GPU Register File using Static Data Compression", *ICPP'20: 49th International Conference on Parallel Processing*, Edmonton, AB, Canada (Virtual Conference). August 2020.

III. **Alexandra Angerd**, Erik Sintorn, and Per Stenström. "GBDI: Going Beyond Base-Delta-Immediate Compression using Global Bases", under submission since December 2020.

IV. **Alexandra Angerd**, Keshav Balasubramanian, and Murali Annavaram. "Distributed Training of Graph Convolutional Networks using Subgraph Approximation", under submission since October 2020.

Part of the contributions resulted in an invention:

- **Alexandra Angerd**, Angelos Arelakis, Erik Sintorn, and Per Stenström. "Systems, Methods and Devices for Exploiting Value Similarity in Computer Memories", P138180009 BA, submitted to Swedish Patent and Registration Office (PRV), December 1st 2020.

Other publications by the author, not included in the thesis:

- **Alexandra Angerd**, Erik Sintorn, and Per Stenström. "A Framework for Automated and Controlled Floating-Point Accuracy Reduction in Graphics Applications on GPUs", *MCC2017: 10th Nordic Workshop on Multi-Core Computing*, Uppsala, Sweden, November 2017.

- **Alexandra Angerd**, Erik Sintorn, and Per Stenström. "A Register File Organization to Support Variable Floating-Point Precision in GPUs", *MCC2018: 11th Nordic Workshop on Multi-Core Computing*, Gothenburg, Sweden, November 2018.

Contents

Acknowledgments

"You can, you should, and if you're brave enough to start, you will". Stephen King, On writing: A Memoir of the Craft. To me, pursuing a PhD is about being brave. It's about being brave enough to meet challenges, hold presentations, be judged, be rejected, speak your mind, try new things, learn new things, believe in yourself. It's about exposing yourself to the unknown, and be brave, until rejections turns into accepts and scary turns into exciting. When I first started on this journey, I thought that would happen in about a year. Now I know better, but now I also hope that I always have to be brave, because I've come to believe that's the most exciting (and rewarding!) way forward.

Luckily, I was not alone in this endeavour. Many people have supported me, in different ways, to tread the path that lead me to finish this thesis.

First, I want to cordially thank my advisor Professor Per Stenström for his immense support throughout these years. He has taught me everything I know about research, for which I am extremely grateful. Per has also supported me in realizing personal goals, such as to go on a research visit, attend summer schools, and challenge my stage fright. Per's passion and contagious enthusiasm for research and innovation have been great motivators for me, and he has truly inspired me to be brave and believe in myself.

I also would like to thank my co-advisor Dr. Erik Sintorn. We have collaborated closely in most of the projects I've been involved in, and I have really enjoyed our sessions of understanding, breaking down, and solving problems (not necessarily in that order). Also, his sense of humour is as "great" as mine, which has elevated many moments.

Further, I want to thank Dr. Lars Svensson for encouraging and inspiring me to apply for a PhD position. I don't believe I would have considered the opportunity without him. Thanks to my examiner Dr. Fredrik Dahlgren, for his support and feedback during the years, and to Professor Murali Annavaram at University of Southern California, for accepting me as a visiting researcher in his group and advising me during my time in Los Angeles.

I am also grateful to all my, past and present, colleagues at Chalmers: Nadja, Christoffer, Lena, Dan, Sverker, Roc, Ulf, Viktor, Mads, Boel, Victor, Erik, Angelos, Madhavan, Bhavi, Fatemeh, Behrooz, Dmitry, Monica, Jan, Rolf, Miquel, Pedro, Andreas, Niklas, Per, and others. Thank you for the great discussions, coffee breaks, lunches, and laughs.

I also want to thank my friends outside of Chalmers, especially Kristina and Mia, who have shared my ups and downs happily and patiently.

Finally, I want to express my deepest gratitude to wonderful family: Sven-Åke, Marie, Catharina, my love and fiancé Calle, and Ångström. They have always been supportive and believed in me through all my decisions in life, even when I have not been so sure myself. For that, I am forever grateful.

This research was funded by the Swedish Research Council (VR-2014-06221 and VR-2019-04929), and The Osher Endowment supported my research visit.

Mölndal, December 2020

Chapter 1

Introduction

Since the beginning of the information age, the amount of created data has grown exponentially. According to IDC's forecast on data growth, the total amount of data created, captured, copied, and consumed worldwide in the year of 2020 is around 59 zettabytes (10^{21} bytes), and is projected to grow to 149 zettabytes by 2024 [18].

This explosion of data is an invaluable source of information, which helps us to learn more about the world. For example, large sets of scientific data let us explore space, make new medical discoveries, predict our own impact on the environment, and find new elementary particles. Also, big data is a fundamental requirement of many new inventions and applications in areas such as AI, the internet of things, the entertainment industry, manufacturing, and healthcare.

An important prerequisite to harness the information embedded within the data is to have enough computing power to process it in a reasonable amount of time. However, while the amount of available information explodes, the future of computational performance growth is uncertain due to a number of barriers in technology. Historically, the increase in computational performance was mainly driven by transistor and clock frequency scaling according to Moore's law and Dennard scaling [10]. However, as Dennard scaling reached the end of the road more than a decade ago due to increased current leakage in the increasingly smaller transistors, architectural innovations based on thread level parallelism (TLP) such as chip multiprocessors and GPUs have paved the path forward.

TLP-based approaches, such as GPUs, also pose challenges. One key challenge is accessing memory fast enough to feed the computational circuits. As technology has progressed, the advances in improvement of memory access time and bandwidth have fallen behind the improvements in computational performance. This is commonly known as the memory wall [43]. For example, GPUs offer a high computational throughput by designating a large part of the die area to computational circuits, which can be individually accessed by parallel threads. The challenge is to balance the use of resources in such a way that the computational fabric is fully utilized.

To reach peak performance, three main conditions, all related to memory, have to be met:

1. Enough threads are available to fill the pipeline and hide memory latency. The number of available threads is limited by the size of the register file and shared memory.

2. Enough data is fed from the device memory to the chip. This is limited by memory bus width and memory speed.

3. The application data fits in device memory. This is, naturally, limited by the size of the main memory.

One important technique towards alleviating the effects of the memory wall is data compression. By reducing the amount of information that needs to be communicated and stored, memory resources crucial for performance can be efficiently utilized. To this end, many compression algorithms directed towards the memory hierarchy have been proposed. The algorithms can be divided into two categories: lossless and lossy, where lossless algorithms are the most important, because loss of information in an uncontrolled way is not acceptable [37].

Lossless techniques aim at compressing data without any loss of information. Lossless compression algorithms are usually either temporal-value based or spatial-value based [37]. In temporal-value based algorithms, frequently occurring values are encoded using smaller code words. Examples include Huffman coding [17], Frequent Value Compression [44], Lempel-Ziv [45], and Run-Length Encoding. In contrast, spatial-value based compression techniques leverage that values tend to cluster around a base value, and that values can be efficiently compressed as small deviations from a base value (e.g. Base-Delta-Immediate (BPC) [32], Bitplane Compression (BPC) [23], and Thesaurus [11]).

In contrast to lossless techniques, a lossy compression algorithm returns an approximated version of the original data. Lossy compression can be applied to applications in which a small deviation of the output is acceptable. This is often the case in domains such as real-time rendering, deep learning training and inference, multimedia, and physics simulations, among others. Some examples of lossy compression techniques include down-sampled textures in graphics applications, precision-reduction of floating-point values [38], quantization of neural networks (e.g. Deep Compression [15]), and storing approximately similar memory blocks [28].

Common for most compression techniques, lossless or lossy, are that they are carried out in one level only of the computer abstraction layer, typically within the microarchitecture, to provide low-latency compression. However, for lossy compression techniques, this results in complications: how much deviation from the exact answer is tolerable must be established at the application-level. Hence, for the produced output to be usable, the lossy compression algorithm has to be aware of how much approximation the application can tolerate through an error bound. At the same time, system- and architectural support is needed to translate the error bound into efficient performance gains. Clearly, lossy compression techniques and cross-layer frameworks aimed at controlling the amount of approximation are required to fully utilize the potential of lossy compression algorithms.

This thesis aims at providing a set of *controlled* compression and approximation techniques for GPUs, with the goal of efficiently utilizing the computational fabric, and thereby increase performance. As will be shown in this thesis, controlled approximation can substantially lower the amount of information the system has to process while providing high output quality, and is thus an important tool in the process of debricking the memory wall.

1.1 Problem Statement

This thesis builds upon the work presented in Papers **I-IV**. The addressed problems aim at exploring approximation as a way of decreasing the amount of processed information in GPU systems, while still returning a result which is sufficiently accurate.

The first problem investigated in this thesis is related to the precision of floating-point values. Reducing the precision of floating-point values by narrowing the bitwidth is an efficient technique to reduce the information needed to be handled by the computer system, and can be leveraged to design systems with higher performance [19] and higher energy-efficiency [20, 39].

However, reducing the floating-point precision in a *controlled* manner, which guarantees that the output of the application stays within an acceptable error bound, needs support across many of the abstraction layers. First, at the application level, the error bound has to be established, which is how much the produced output is allowed to deviate from the exact output. Next, at the assembly language level, each floating-point value has to be annotated with a precision which guarantees the error bound. At last, compiler and architectural support is needed to translate the floating-point annotations into efficient resource utilization. To this end, the thesis attempts to answer the following questions: How can a connection, which guarantees that the output is within a certain specified error bound, be established across the computer abstraction levels? Next, if such a connection can be established, how can it be leveraged to design microarchitecture features which translate approximations into increased performance? These questions are investigated in Papers **I** and **II**.

Another problem, which is explored in paper **III**, involves compression of floating-point values. While precision reduction is efficient in decreasing processed information, it does not exploit the dimension of value locality which is often present in sets of data [37]. However, floating-point values are known to be more challenging to compress than other data types, for example integers, due to how they are stored in bit space [2]. Thus, questions arise: how can floating-point values be compressed efficiently, and how can approximations further increase compression while keeping the output error low?

Finally, this thesis also seeks answers to a problem related to processing of very large datasets, common in the domain of training deep neural networks. If application data do not fit in GPU device memory, the system has to wait for data to be loaded from elsewhere (storage or CPU RAM). Since the data is not readily available, this results in performance degradation. A common solution for applications such as Convolutional Neural Networks (CNNs) is to distribute the training across a cluster of multiple GPU-equipped machines. This is done by dividing the dataset into a number of chunks, and let each machine train a local model on its assigned data chunk. A single global model can be constructed by aggregating the model parameters across the machines [9].

Although this approach is efficient for CNNs, that may operate on independent pieces of input data such as images, data-parallel distributed training is more challenging for other types of neural networks. For example, recent efforts in research have adapted CNNs to operate directly on data modelled as graphs [5, 14, 13]. This type of neural network is called Graph Convolutional Networks (GCNs). However, splitting a large graph into smaller, independent subgraphs requires removing the edges between the subgraphs, which causes accuracy problems since information about the data is lost. To this end, Paper **IV** seeks the answer to the question: is it possible to adapt data-parallel distributed training to GCNs while maintaining a high accuracy?

1.2 Thesis Contributions

This thesis considers three areas: controlled floating-point precision reduction, lossless and lossy memory compression, and distributed training of GCNs.

In the first area, this thesis shows that through automated and controlled floating-point approximation, the number of threads available to the GPU computation units can be increased by efficiently utilizing the register file resources. This is achieved through two contributions:

- A heuristic for automating the process of selecting the precision of each floating-point value given a certain output quality threshold. The worst-case complexity of the heuristic is bounded by $O(an)$ as opposed to $O(an^2)$ in previous work [36], where a is the number of floating-point precision levels and n is the number of analyzed values (Paper **I**).

- A novel register file organization and microarchitectural implementation (Paper **II**) together with a register file allocation algorithm (Paper **I**) that together can leverage a lower precision of floating-point numbers, as well as narrow integers, for increased capacity and performance. Evaluations show that up to twice as many threads can be active simultaneously which translates to a performance improvement of up to 79%, 18.6% on average, when allowing for a slight output quality loss.

Within the second area, Paper **III** aims at increasing the effective bandwidth of GPUs by presenting a lossless and lossy memory compression algorithm to reduce the amount of transferred data. While state-of-the-art techniques such as BDI and BPC build on the hypothesis that intra-block values are similar and can be encoded efficiently using an intra-block base value, Paper **III** makes the observation that this is not always true. Rather, values tend to cluster around a small number of global base values. Based on this observation, Paper **III** makes the following contributions:

- A lossless compression scheme, GBDI, for read-only data which uses multiple global base values to achieve a higher compression ratio than BPC. The evaluation shows that lossless GBDI offers a substantially higher geometric mean compression ratio (3.34x) than BPC (2.51x) across both integer and float benchmarks.

- A new controlled approximation technique for floating-point values that offers even higher compression ratio than lossless GBDI for floating-point benchmarks, if controlled approximation is allowed. At a given error rate, this technique offer a higher compression ratio than mantissa truncation.

Finally, Paper **IV** makes contributions in the third area. It explores GCN training in a distributed setting, and observes that training on mutually exclusive subgraphs results in a substantially lower accuracy than what full-graph training on a single machine would yield. To mitigate accuracy loss in a distributed setting, Paper **IV** makes the following contribution:

- A subgraph approximation scheme for graph data which mitigates the accuracy loss and allows training of GCNs in a distributed setting to converge at single-machine accuracy, while minimizing synchronization overhead between the machines.

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 provides brief summaries of Papers **I-IV**. Chapter 3 makes concluding remarks and outlines future work.

Chapter 2

Summary of Papers

This chapter provides summaries of Papers **I** to **IV**. All papers are provided as appendices to the thesis.

2.1 Paper I

This section provides an extended abstract of Paper **I** titled "A Framework for Automated and Controlled Floating-Point Accuracy Reduction in Graphics Applications on GPUs" and published in *ACM Transactions on Architecture and Code Optimization (TACO), Volume 14 Issue 4, December 2017.*

2.1.1 Summary

Reducing the precision of floating-point values is a technique which can be used to both achieve higher performance [19] and higher energy-efficiency [20, 39]. This is especially true for GPUs, since many of its common tasks, for example image-rendering, are in domains in which the output is inherently insensitive to precision-reduction [33]. Since values with low floating-point precision can be expressed with fewer bits, a substantially lower precision can open up for many optimization approaches such as more resource-efficient register files [26], data paths [38], functional units [31], and cache subsystems [19]. In order to leverage microarchitectural support for narrow floating-point values, the operands have to be annotated with a suitable precision at the instruction level.

The aim of Paper **I** is to propose a framework which provides instruction-level annotations in an automated and controlled manner. Automated means that the framework automatically, and transparently to the programmer, sets an appropriate precision for each floating-point value in the program. Controlled refers to guaranteeing that the quality of the output does not undercut a certain quality threshold which is defined by the programmer at the application level.

Previous efforts have been made to provide a framework for automated precision tuning. For example, Precimonius [35] automatically selects among IEEE754-compliant data types for each source-level floating-point variable. However, the approximation levels in Precimonius are constrained to IEEE754-compliant data types, and the optimization targets source-level variables rather than instruction-level floating-point operands. Furthermore, the scalability of the algorithm is limited ($\mathcal{O}(pn^2)$, where p is the number of precision levels and n is the number of floats to tune).

Another work which targets automated precision tuning is Chisel [29], but like Precimonius, it does not target instruction-level floats. In addition, Chisel employs a static interval analysis, which gives an upper bound on the required precision, resulting in conservative bitwidths.

Unlike previous work, Paper **I** presents a data-driven method for automated precision tuning which operates directly on compiled and optimized assembly code. The proposed algorithm has a lower computational complexity than Precimonius, with a worst-case computational complexity bounded by $\mathcal{O}(pn)$. This is important, since Paper **I** targets virtual registers in Single Static Assignment (SSA) form, of which there can be hundreds or thousands. The goal of the proposed algorithm is to identify which floating-point registers can be represented by lower precision formats and how much the precision can be lowered, while still meeting the specified quality threshold. To find the optimal set of all values and all precisions, an exhaustive search is required. Since such a solution would not be practically viable (with a complexity of ($\mathcal{O}(p^n)$), Paper **I** proposes a heuristic algorithm which empirically returns good results.

To show the benefits of floats with reduced precision, Paper **I** also sketches the specification of a register file organization with the capability of storing floats of variable precision densely. The register file is of particular interest in GPU architectures. This is because GPUs hide latency by keeping a large number of threads in flight simultaneously, and schedule a new thread whenever a running thread stalls on a memory operation. Context switches happen at the cycle level, which means that each active thread has to keep its state readily available in the register file until it terminates. Hence, the size of the register file has a direct impact on how many threads can be active simultaneously, and thus affects performance.

The proposed register file organization stores variable-precision values densely by dividing each physical register into slices, and allocate as many slices as needed for each value. This makes it possible to store multiple values within the same physical registers. The slices are accessed through an indirection table, where pointers to the slices are stored for each value. Since the size of each value is known at compile time, the indirection table can be configured by the register allocator at compile time. To bridge the gap between the value annotations and the register file, Paper **I** also proposes a register allocation algorithm which is aware of the width of each value.

The contributions are evaluated on five graphics kernels. First, they are compiled into a hardware-independent assembly format (LLVM IR [25]). Then, instructions are injected that make it possible to dynamically set the precision of each floating-point value. Next, the proposed precision-tuning heuristic is applied to annotate each floating-point with a precision. The annotations are then used by the width-aware register allocation algorithm to place the values into the sliced register file. The output from the register allocation step corresponds to the *register pressure*, that is, how many register resources are used by each thread.

The evaluation compares three different low-precision floating-point formats: IEEE754-compliant, mantissa truncation, and IEEE754-style. The IEEE754-compliant format considers the approximation levels provided by the IEEE754 standard (32-bit single-precision , and 16-bit half-precision). The mantissa truncation format assumes a single-precision floating-point value where the mantissa bits are dropped to a lower precision, but the exponent bits are kept as is. The IEEE754-style format dynamically decreases the width of both the exponent and mantissa fields, but keeps the ratio between these fields approximately the same as in the IEEE754 standard. For both the mantissa truncation and IEEE754-style format, seven precision levels are considered (32 to 8 bits, in steps of 2).

The results show that the fine-grained formats, IEEE754-style and mantissa truncation, enable the framework to remove more bits than the IEEE754-compliant format does, at the same quality threshold. Furthermore, IEEE754-style outperforms mantissa truncation in terms of removed bits.

In summary, if the IEEE754-style format is used, the register pressure can be reduced up to 48%, 27% on average, while still maintaining a high output quality. The results indicate that it is possible to considerably increase the number of in-flight threads in GPUs by accepting a small loss of output quality, and use the quality threshold as a knob to control the number of threads in flight.

2.2 Paper II

This section provides an extended abstract of Paper **II** titled ”A GPU Register File using Static Data Compression” and published in the proceedings of *ICPP’20: 49th International Conference of Parallel Processing, Edmonton, AB, Canada. August 2020.*

2.2.1 Summary

As mentioned in Section 2.1, GPUs use large register files to achieve high throughput and enable latency hiding of memory accesses. The trend is to rely on increasingly larger register files in order to further increase thread-level parallelism (TLP) [30]. However, large register files are power hungry, which makes it important to seek for new approaches to improve their utilization.

Paper **I** showed that precision-reduction of floating-point values has the potential of substantially lower the register pressure of each thread, while still maintaining a high-quality output. However, Paper **I** only assumes the existence of a register file with support for low-precision floats and does not present any microarchitectural design of such a register file, nor how it could be integrated into a GPU pipeline. Additionally, Paper **I** does not evaluate the potential performance improvement given by such a register file.

Furthermore, Paper **I** does not investigate any other data types than floats. However, studies have shown that a large portion of the integer operands stored in the GPU register file are *narrow* [12, 41], meaning that they require less than the width of a standard register of 32 bits. Support for narrow integers could further lower the register pressure of each thread.

Previous work in the area [12, 41, 3] propose register file organizations which pack operands by establishing their bitwidth at run time. This is achieved by using zero/one-detection logic to remove redundant sign bits. However, this approach does not work for floating-point data. In addition, the precision of floating-point operands cannot be decided at run time, since it is not possible to know what impact the precision-reduction of a specific operand would have on the end result.

Paper **II** proposes a register file organization capable of storing operands at a fine granularity regardless of operand type. This is achieved through a compiler-assisted approach which annotates the precision needed for each operand at compile time. Narrow integers are detected by leveraging static range analysis [34], while the precision of each float is determined using the framework presented in Paper **I**. The annotations are taken into consideration to achieve a dense register allocation using the allocation algorithm from Paper **I**. At run time, the proposed register file uses a configurable indirection table to store the location of each operand.

The design of the proposed register file implies three main challenges, which are addressed in Paper **II**. First, the indirection table is on the critical path, since it has to be consulted for each register lookup. Hence, it is important that the latency of the indirection table is low. Second, the indirection table also has to handle multiple accesses per cycle, since several registers have to be read in parallel. Third, the proposed register file is designed to support the IEEE754-style format presented in Paper **I**, which means that conversions has to be carried out between full-precision floats and the low-precision formats which is used to densely store the floats in the register file. Paper **II** mitigates these challenges by presenting an indirection table which matches the throughput of the register file, as well as conversion units capable of carrying out the float conversion in one processor cycle.

The proposed design is evaluated by modifying the cycle-accurate GPU simulator GPGPU-Sim [4] to include the added microarchitectural components. GPGPU-Sim simulates NVIDIA's instruction set PTX, so the framework from Paper **I** is modified to annotate each PTX register with a bitwidth. The framework outputs the indirection table contents which points to the location of each operand, which is then uploaded to GPGPU-Sim. In practise, a small ISA extension could be employed to load the data into the indirection table. This needs to be carried out only once per kernel launch, since the register allocation is static.

The extended GPU pipeline is evaluated on eleven GPU kernels from various application domains, in which the theoretical number of threads in flight is limited by register pressure. The first four kernels are the graphics kernels evaluated in Paper **I**. The other kernels are from benchmarks in the Rodinia benchmark suite [6].

To tune the float precision, each kernel is coupled with a quality metric suitable for its output. The graphics kernels output images and uses Structural Similarity Index [42] as metric, while the Rodinia benchmarks use either percentage of deviation from the correct output (six kernels) or a binary metric of either correct or wrong (one kernel). Note that the choice of quality metric has a large impact on how efficiently the framework can remove bits, as well as the usability of the output. Ideally, the quality metric should be established by domain experts. In Paper **II**, the coupled quality metrics are used to show the potential of the work. Other quality metrics could be used without any other change in methodology.

The kernels are evaluated in terms of register pressure, *occupancy* (the ratio of active threads to the maximum number of threads supported by the core), and performance in terms of Instructions Per Clock (IPC). The results show that it is necessary to consider both narrow integers and precision-reduced floats to achieve a substantial register pressure reduction across all benchmarks. This register pressure reduction increases the occupancy, which leads to an IPC increase by up to 79%, 18.6% on average, when allowing for a slight output quality degradation. While register pressure reduction leads to an average increase in IPC, the results also show that this is not always the case. It is important to note that, while TLP is a prerequisite for high performance in GPUs, a higher occupancy does not always lead to increased performance due to the overall system complexity [40]. For example, the pipeline might already be fully utilized, or the increased number of threads might cause contention in caches.

In addition to the performance evaluation, Paper **II** also presents area and power overhead analyses. The area overhead analysis uses transistor count as a proxy to estimate area of the added microarchitectural structures and finds that the total area increase is less than 1% of the transistor budget of the baseline architecture. The power overhead is estimated analytically, and shows that the proposed design increases power less than what a twice as big a register file would do. This comparison is interesting since for some kernels, the proposed design more than double the number of active thread blocks.

In summary, Paper **II** proposes a new concept for efficient register-packing, which combines static integer and float compression with a novel GPU register file organization capable of lowering the register pressure. It presents a microarchitectural implementation of the proposed organization, together with an evaluation as well as overhead estimations for area and power. The results show that the IPC of the investigated kernels can be increased by up to 79%, 18.6% on average, when allowing for a slight output quality degradation.

2.3 Paper III

This section provides an extended abstract of Paper **III** titled "GBDI: Going Beyond Base-Delta-Immediate Compression using Global Bases". At the time of this writing, the paper is under submission.

2.3.1 Summary

The performance of many GPU applications are limited by the available memory bandwidth [40]. One approach to make more bandwidth available is to employ compression of the transmitted data, which increases the amount of information that can be communicated each clock cycle.

While many general compression techniques exist [37], compression in the memory hierarchy has to incur a low latency since memory reads are on the critical path. Thus, compression algorithms tailored for the memory hierarchy are typically less complex than algorithms targeted for purposes such as storage.

One recently proposed compression technique to increase bandwidth in GPUs is Bitplane Compression (BPC) [23]. It is built upon Base-Delta-Immediate (BDI) compression [32], in which a block of data values is compressed by utilizing data similarity within the block. This is done by selecting one of the values as a base, and encode the other values as a difference (*delta*) from that base.

The effectiveness of BDI-based compression relies on the hypothesis that the delta can be compactly encoded because the values within the block are similar. However, Paper **III** makes the observation that this is not always the case. This is especially true for floating-point values, since the IEEE754 standard, where the bits in a value is split into three regions (sign, exponent, and mantissa) results in that numbers that are perceived as numerically similar cannot always be encoded compactly by using deltas.

Paper **III** observes that values tend to cluster around a few base values, and these that clusters are spread across multiple value blocks. Based on this observation, the paper proposes a lossless compression algorithm for read-only data, called GBDI, that uses *global bases* shared by all input data, instead of intra-block bases as in BDI. GBDI compresses the read-only data offline, and decompresses the data at runtime.

The idea of GBDI is to use a small number of global bases across blocks to minimize deltas, where each value is compressed as a global base pointer and a delta. The global base values are stored in a small indexed lookup table. GBDI takes a block of values as an input. For each value, it locates the closest global base value, and computes the delta. Then, for each value, the delta and global base pointer are concatenated into one data array. These data arrays are then further compressed using the same approach as BPC: the matrix consisting of the data arrays are transposed such that each row represents one bit position of each data array, called *bitplanes*. These bitplanes are compared against a frequent pattern table, and encoded with a small pattern representation if a match is found.

The main challenge of GBDI is to find suitable global base values. An intuitive approach is to locate value clusters through a standard clustering technique such as kmeans [27]. However, such clustering techniques usually minimize the *distance* between the cluster center (base) and the values. Paper **III** makes the observation that this approach is suboptimal for BDI-based compression. Rather, what should be minimized is the *number of bits* needed to encode the distance. Paper **III** shows that finding an optimal base b among M values in a set of values $x_i \in X$ is equivalent to minimizing the function $f(b) = \sum_{i=0}^{M-1} log_2(1 + |b - x_i|)$, and that the global minimum is in one of the values x_i.

Kmeans can be modified to minimize the function $f(b)$ instead of the distance, which substantially increases the compression ratio of GBDI when the cluster centers are used as global bases. However, this solution is computationally heavy, since $f(b)$ has to be evaluated for all x_i to locate the global minimum. Paper **III** employs a subsampling scheme to reduce the complexity, but the complexity is still high ($\mathcal{O}(M^{3/4})$). Therefore, a heuristic with lower complexity ($\mathcal{O}(Mlog(M))$), called *histogram binning*, is also proposed.

The goal of histogram binning is to ensure that a majority of values are close to a base value. This is achieved by constructing a histogram of all values, and split it into a number of equal-range bins. The bins that contain the most values are identified, and from these bins, the most frequently occurring value is selected as a base. To find suitable bases, the heuristic sweeps over a fixed number of bases and bins, compresses the data with GBDI, and chooses the combination which maximizes compression ratio.

In addition to the lossless compression with global bases, Paper **III** also proposes a lossy extension to GBDI which allows the algorithm to leverage approximation of floating-point values. This extension is called BitPlane-Aware Approximation (BPA), and is applied to floating-point values before the deltas are computed. Essentially, BPA analyzes the t least-significant mantissa bitplanes of the floating-point values, where t is an approximation threshold set by the user. It approximates the bitplanes to either 0 or 1, depending on which one is most common, and stores the approximated bitplanes as a small array. During decompression, the t-bit array are inserted into the reconstructed values to restore an approximate version of the original values. Compared to truncation, when the t least significant bits are simply set to 0, BPA offers a higher compression ratio at a fixed, low, error rate.

To evaluate GBDI, GPU kernels from the Rodinia [6] and Parboil [1] benchmark suits are investigated. The CUDA memory copy call feeds input data into GBDI, and the resulting compression ratio is reported. The compression ratio is analyzed as a function of the size of the global base lookup table, when histogram binning is used to locate global bases. The results show that a global base table size of only 256B (64 32-bit bases) has a geometric mean compression ratio, across all benchmarks, of 3.34X, compared to 2.51X for BPC using intra-block bases. In summary, for most benchmarks, GBDI yields a substantial increase in compression ratio as compared to BPC. The exception is if the benchmark contains values which gradually increase as we sweep over the memory. In this case, an intra-block base yield very small deltas and thus a high compression ratio.

The results also compare unmodified and modified kmeans (using distance and $f(b)$ as minimization objective, respectively). The modified version performs better with a total geometric mean of 3.10X as compared to 2.60X. However, histogram binning achieves an even higher compression ratio for many benchmarks (total geometric mean: 3.34X). This is because histogram binning continuously evaluates the compression ratio, and chooses the best performing bases. Since compression ratio not only depends on the size of the deltas, but also the values within each block, histogram binning takes more dimensions into consideration and might find better global bases.

In summary, Paper **III** contributes with a lossless compression algorithm for read-only data, which uses multiple global bases to achieve a higher compression ratio than BPC. It also shows that choosing base values that minimize the distance between cluster values does not maximize compression, and proposes to use the number of bits encoding the distance instead. Furthermore, the paper contributes with histogram binning for clustering, a heuristic which yields higher compression ratio than kmeans. Finally, the paper contributes with a new approximation technique for floating point values which can further increase compression ratio.

2.4 Paper IV

This section provides an extended abstract of Paper **IV** titled "Distributed Training of Graph Convolutional Networks using Subgraph Approximation". At the time of this writing, the paper is under submission.

2.4.1 Summary

Data modeled as graphs are ubiquitous in application domains such as social networks, biological networks, and e-commerce interactions. Recent research has adapted machine learning techniques to train directly on graphs. One important breakthrough is the development of Graph Convolutional Networks (GCNs), which generalizes the function of Convolutional Neural Networks (CNNs), to operate on graphs [24].

However, many real-world graphs are typically very large and do not fit in GPU device memory, often making the problem of training machine learning models on them intractable. GCNs are typically more memory demanding than CNNs since processing a single vertex requires accessing its neighbours, which means a large amount of data have to be accessed if the neighborhood is large. Additionally, GCN algorithms often access neighbors multiple hops away, which might require traversal of disparate graph sections without locality. This poses challenges when the graph does not fit in memory. Previous approaches alleviate these issues by constraining the number of dependent vertices through sampling-based methods [13, 7, 16, 8]. However, heavily sampling the graph may limit the achievable accuracy [21].

To alleviate the memory size constraints and the computing capacity of a single machine, the training of CNN models turned to data parallel distributed training. In such training, a number of machines independently train on identical neural networks, but on different chunks of input data. A single global model can be constructed by aggregating model parameters at regular intervals, for example at the completion of a batch [9]. This works well for CNNs because the input data can be divided into independent chunks. For example, if the input is a set of images, each image is independent. Hence, when the dataset is split into chunks, no information is carried between the chunks and each chunk can be trained on independently. In contrast, if the input is a connected graph, the equivalent of splitting the dataset into chunks is to partition the graph into a number of subgraphs. Since the training of GCNs require not only the vertex information but also the neighborhood, the GCN might need information located on a different machine. If all such cross-machine information would be communicated, a communication bottleneck would quickly arise.

Another approach is to ignore the information that spans multiple machines [21]. However, this solution might result in accuracy loss. For example, when distributed on five compute nodes and ignoring cross-machine edges, Paper **IV** notes that the accuracy of GraphSAGE [13] trained on the Reddit dataset [13] drops from 94.94% to 89.95%, a substantial loss.

To mitigate this problem, Paper **IV** proposes a distributed training approach for GCNs which reaches an accuracy comparable with that of training on a single-system machine, while keeping communcation overhead low. This allows GCNs to train on very large graphs on a distributed system where each machine has limited memory. This is achieved by a proposed subsampling scheme, called *Subgraph Approximation*, which approximates the non-local subgraph and making the approximations available in the local machine. The intuition behind Subgraph Approximation is that each machine has a complete view of one subgraph, and an approximate view of the rest of the subgraphs.

The scheme first partitions the graph into n non-overlapping subgraphs $g_i \in G$, which are roughly equal in size. For each subgraph g_i a fixed number of vertices are randomly sampled for each of the other subgraphs in G, if the vertex has a path to g_i. Next, the subgraph g_i is extended to include the sampled vertices v_s, including the edges between v_s and g_i. The number of sampled vertices is a user-defined input parameter.

The approach is evaluated on a cluster of identical machines, each equipped with an AMD Ryzen 3 1200 Quad-Core processor, 32 GB memory, and a NVIDIA GeForce GTX 1080 Ti. The cluster is configured in a master-worker setting, where the master acts as a parameter server, and the workers are responsible for the training. The master initiates the training by partitioning the graph using the METIS library [22], applying subgraph approximation, and distributing the subgraphs among the workers. The workers send their trained model parameters to the master at the end of every epoch. The master aggregates the parameters and averages them before distributing the results to the workers, which initiates the next epoch. At the end of the training period, the aggregated parameters from all workers form a global model.

The approach is evaluated on two GCNs: the GCN proposed by Kipf and Welling [24] (called KW-GCN in the paper), and GraphSAGE [13], and two datasets: Reddit [13] with 231K vertices and 11M edges, and Amazon2M [8] with 2M vertices and 48M edges.

The evaluation first explores distributed training for the two GCNs when trained on the Reddit dataset. On a single machine, the training for KW-GCN and GraphSAGE converges at 92.90% and 94.94%, respectively. However, when training on 5 workers in a distributed setting *without* Subgraph Approximation, the corresponding accuracy is 50.20% and 89.95%, respectively, for KW-GCN and GraphSAGE. Paper **IV** further investigates Subgraph Approximation by trying different numbers of sampled vertices. For GraphSAGE, an overhead of 2% of the *total* vertices, added to each machine, is enough to reach single-machine accuracy. A further increase in overhead lets the model converge faster, but does not necessarily increase accuracy.

KW-GCN is more sensitive to graph partitioning than what GraphSAGE is. However, adding a small overhead substantially increases the model accuracy bringing it close to the baseline (91.80% for an overhead of 5%).

When it comes to Amazon2M, it is more challenging to train on a single machine for both GCNs: for KW-GCN, the adjacency matrix is too large to fit in the GPU. For GraphSAGE, the training time is very long. However, the trends are similar to that of training on the Reddit dataset: a small increase in overhead substantially increases both accuracy and time to convergence.

In summary, distributed GCN training is efficient in reducing execution time and memory footprint, but it can also cause accuracy problems. Paper **IV** proposes a technique to mitigate the accuracy loss by introducing subgraph approximation which helps the system to reach single-machine accuracy while still keeping the memory footprint and communication costs low.

Chapter 3

Concluding Remarks and Future Work

As the amount of available information grows, new innovations in computing devices are needed in order to process the information in a reasonable amount of time. One key challenge is to overcome the memory wall, which has arisen because the improvements in memory access time and bandwidth have fallen behind the improvements in computational performance.

To this end, this thesis proposes a number of compression and approximation techniques for GPUs, with the goal of efficiently utilizing the computational fabric. The results show that the techniques presented in this thesis have the potential of substantially increasing the performance of GPUs, especially if a slight output quality degradation is allowed.

One contribution of this thesis is a framework which enables floating-point precision in a controlled manner, which guarantees that the output is above a pre-specified quality threshold. While the tuning method presented in Paper **I** is applicable to any application, one challenge is to establish a *quality* metric for all applications. This is not the same as an error measurement, since an error measurement does not say anything about the end users' perception of the output, nor how usable the output is. One direction of future work is to establish well-defined quality metrics for more domains in order to efficiently apply the proposed precision-tuning method.

Moreover, this thesis proposes to leverage the precision-annotated instructions returned from the precision-tuning method to optimize the register file. However, many other optimization opportunities are possible. For example, annotations for load- and store instructions indicate that less data could be read from and written to memory, which in turn may open up for further optimizations in other levels of the memory hierarchy. On a similar note, the annotations could be used as a guidance when deciding on the approximation threshold for BitPlane-Aware Approximation, as presented in Paper **III**.

One limitation of GBDI, the compression algorithm presented in Paper **III**, is that it only targets read-only data. This is because an offline analysis of all values is needed in order to locate suitable global base values. However, another work in the area of compression techniques applied to the memory hierarchy has made the observation that value locality changes slowly, and has proposed statistical compression schemes which only periodically profiles the data [2]. An interesting future research direction is to investigate if it is possible to extend GBDI for read-write data by employing an approach in the same spirit. Furthermore, while Paper **III** analyzes the possible compression ratio of GBDI, it does not explore what impact the compression scheme has on the available bandwidth and performance. This is also a target for future studies.

When it comes to Paper **IV**, an interesting direction of future work is to investigate subgraph approximation for even larger distributed systems and graphs. Also, the current study only explores distributed *training*, not distributed *inference*. Hence, the study could be extended by also exploring if subgraph approximation is viable in an inference setting. Another possible direction is to investigate load-balancing in combination with subgraph approximation. While it is relatively easy to partition a graph into roughly equal subgraphs in terms of vertices, the time it takes to train on one subgraph also depends on the size of the neighborhood for each vertex. Hence, if the subgraphs are equal in terms of vertex count but not in sparsity, the load might not be balanced which increases execution time of the distributed system. Finally, Subgraph Approximation uses randomized sampling to approximate non-local subgraphs. However, deterministic sampling such as selecting the vertices with the highest number of edges to the local subgraph might perform better, since the information from such vertices are beneficial to many vertices in the local subgraph. Exploring other sampling methods is also a possible future research direction.

References

[1] A. Stratton, et al. *Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Programming*. Tech. rep. IMPACT-12-01. University of Illinois, at Urbana-Champaign, 2012.

[2] Angelos Arelakis, Fredrik Dahlgren, and Per Stenstrom. "HyComp: A Hybrid Cache Compression Method for Selection of Data-Type-Specific Compression Methods". In: *Proceedings of the 48th International Symposium on Microarchitecture*. MICRO-48. Waikiki, Hawaii: Association for Computing Machinery, 2015, pp. 38–49. ISBN: 9781450340342. DOI: 10.1145/2830772.2830823. URL: https://doi.org/10.1145/2830772.2830823.

[3] Hodjat Asghari Esfeden et al. "CORF: Coalescing Operand Register File for GPUs". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 701–714. ISBN: 9781450362405. DOI: 10.1145/3297858.3304026. URL: https://doi.org/10.1145/3297858.3304026.

[4] A. Bakhoda et al. "Analyzing CUDA workloads using a detailed GPU simulator". In: *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. Apr. 2009, pp. 163–174. DOI: 10.1109/ISPASS.2009.4919648.

[5] Joan Bruna et al. "Spectral Networks and Locally Connected Networks on Graphs". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014.

[6] S. Che et al. "Rodinia: A benchmark suite for heterogeneous computing". In: *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 2009, pp. 44–54. DOI: 10.1109/IISWC.2009.5306797.

[7] Jie Chen, Tengfei Ma, and Cao Xiao. "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rytstxWAW.

[8] Wei-Lin Chiang et al. "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for

Computing Machinery, 2019, pp. 257–266. ISBN: 9781450362016. DOI: 10.1145/3292500.3330925. URL: https://doi.org/10.1145/3292500.3330925.

[9] Jeffrey Dean et al. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012, pp. 1223–1231.

[10] R. H. Dennard et al. "Design of ion-implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511.

[11] Amin Ghasemazar, Prashant Nair, and Mieszko Lis. "Thesaurus: Efficient Cache Compression via Dynamic Clustering". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 527–540. ISBN: 9781450371025. DOI: 10.1145/3373376.3378518. URL: https://doi.org/10.1145/3373376.3378518.

[12] S. Z. Gilani, N. S. Kim, and M. J. Schulte. "Power-efficient computing for compute-intensive GPGPU applications". In: *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. Feb. 2013, pp. 330–341. DOI: 10.1109/HPCA.2013.6522330.

[13] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1025–1035. ISBN: 978-1-5108-6096-4.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. "Representation Learning on Graphs: Methods and Applications". In: *IEEE Data Eng. Bull.* 40 (2017), pp. 52–74.

[15] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV].

[16] Wenbing Huang et al. "Adaptive Sampling Towards Fast Graph Representation Learning". In: *Advances in Neural Information Processing Systems (NIPS)*. 2018.

[17] D. A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101. DOI: 10.1109/JRPROC.1952.273898.

[18] IDC and Statista. *Volume of data/information created worldwide from 2010 to 2024 (in zettabytes)*. 2020. URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (visited on 11/10/2020).

[19] A. Jain et al. "Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Oct. 2016, pp. 1–13. DOI: 10.1109/MICRO.2016.7783744.

[20] D. Jeong et al. "An eDRAM-Based Approximate Register File for GPUs". In: *IEEE Design Test* 33.1 (Feb. 2016), pp. 23–31. ISSN: 2168-2356. DOI: 10.1109/MDAT.2015.2500185.

[21] Zhihao Jia et al. "Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc". In: *Proceedings of Machine Learning and Systems 2020*. 2020, pp. 187–198.

[22] George Karypis and Vipin Kumar. *METIS—A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Ordering of Sparse Matrices*. Tech. rep. Jan. 1997.

[23] Jungrae Kim et al. "Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures". In: *Proceedings of the 43rd International Symposium on Computer Architecture*. ISCA '16. Seoul, Republic of Korea: IEEE Press, 2016, pp. 329–340. ISBN: 9781467389471. DOI: 10.1109/ISCA.2016.37. URL: https://doi.org/10.1109/ISCA.2016.37.

[24] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[25] Chris Lattner and Vikram Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In: *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*. CGO '04. Palo Alto, California: IEEE Computer Society, 2004, pp. 75–. ISBN: 0-7695-2102-9. URL: http://dl.acm.org/citation.cfm?id=977395.977673.

[26] Sangpil Lee et al. "Warped-compression: Enabling Power Efficient GPUs Through Register Compression". In: *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*. ISCA '15. Portland, Oregon: ACM, 2015, pp. 502–514. ISBN: 978-1-4503-3402-0. DOI: 10.1145/2749469.2750417.

[27] S. Lloyd. "Least Squares Quantization in PCM". In: *IEEE Trans. Inf. Theor.* 28.2 (Sept. 2006), pp. 129–137. ISSN: 0018-9448. DOI: 10.1109/TIT.1982.1056489. URL: https://doi.org/10.1109/TIT.1982.1056489.

[28] J. S. Miguel et al. "The Bunker Cache for spatio-value approximation". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2016, pp. 1–12. DOI: 10.1109/MICRO.2016.7783746.

[29] Sasa Misailovic et al. "Chisel: Reliability- and Accuracy-aware Optimization of Approximate Computational Kernels". In: *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA '14. Portland, Oregon, USA: ACM, 2014, pp. 309–328. ISBN: 978-1-4503-2585-1. DOI: 10.1145/2660193.2660231. URL: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/2660193.2660231.

[30] S. Mittal. “A Survey of Techniques for Architecting and Managing GPU Register File”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.1 (Jan. 2017), pp. 16–28. ISSN: 1045-9219. DOI: 10.1109/TPDS.2016.2546249.

[31] NVIDIA. *NVIDIA Tesla P100*. White Paper. NVIDIA, 2016.

[32] Gennady Pekhimenko et al. “Base-Delta-Immediate Compression: Practical Data Compression for on-Chip Caches”. In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. PACT ’12. Minneapolis, Minnesota, USA: Association for Computing Machinery, 2012, pp. 377–388. ISBN: 9781450311823. DOI: 10.1145/2370816.2370870. URL: https://doi.org/10.1145/2370816.2370870.

[33] Jeff Pool, Anselmo Lastra, and Montek Singh. “Precision Selection for Energy-efficient Pixel Shaders”. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG ’11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 159–168. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018349. URL: http://doi.acm.org/10.1145/2018323.2018349.

[34] R. E. Rodrigues, V. H. Sperle Campos, and F. M. Quintão Pereira. “A fast and low-overhead technique to secure programs against integer overflows”. In: *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2013, pp. 1–11. DOI: 10.1109/CGO.2013.6494996.

[35] Cindy Rubio-González et al. “Precimonious: Tuning Assistant for Floating-point Precision”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC ’13. Denver, Colorado: ACM, 2013, 27:1–27:12. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503296. URL: http://doi.acm.org/10.1145/2503210.2503296.

[36] Cindy Rubio-González et al. “Precimonius: Tuning assistant for floating-point precision”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis, SC* (Nov. 2013). DOI: 10.1145/2503210.2503296.

[37] S. Sardashti et al. *A Primer on Compression in the Memory Hierarchy*. Synthesis Lectures on Computer Science. Morgan & Claypool, 2015. DOI: 10.2200/S00683ED1V01Y201511CAC036.

[38] Vijay Sathish, Michael J. Schulte, and Nam Sung Kim. “Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads”. In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. PACT ’12. Minneapolis, Minnesota, USA: Association for Computing Machinery, 2012, pp. 325–334. ISBN: 9781450311823. DOI: 10.1145/2370816.2370864. URL: https://doi.org/10.1145/2370816.2370864.

[39] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar. “Reducing power by optimizing the necessary precision/range of floating-point arithmetic”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8.3 (June 2000), pp. 273–286. ISSN: 1063-8210. DOI: 10.1109/92.845894.

[40] Vasily Volkov. “Understanding latency hiding on gpus”. PhD thesis. UC Berkeley, 2016.

[41] X. Wang and W. Zhang. “GPU Register Packing: Dynamically Exploiting Narrow-Width Operands to Improve Performance”. In: *2017 IEEE Trustcom/BigDataSE/ICESS*. Aug. 2017, pp. 745–752. DOI: `10.1109/Trustcom/BigDataSE/ICESS.2017.308`.

[42] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4 (Apr. 2004), pp. 600–612. ISSN: 1057-7149. DOI: `10.1109/TIP.2003.819861`.

[43] Wm. A. Wulf and Sally A. McKee. “Hitting the Memory Wall: Implications of the Obvious”. In: *SIGARCH Comput. Archit. News* 23.1 (Mar. 1995), pp. 20–24. ISSN: 0163-5964. DOI: `10.1145/216585.216588`. URL: `https://doi.org/10.1145/216585.216588`.

[44] Youtao Zhang, Jun Yang, and Rajiv Gupta. “Frequent Value Locality and Value-Centric Data Cache Design”. In: *SIGOPS Oper. Syst. Rev.* 34.5 (Nov. 2000), pp. 150–159. ISSN: 0163-5980. DOI: `10.1145/384264.379235`. URL: `https://doi.org/10.1145/384264.379235`.

[45] J. Ziv and A. Lempel. “Compression of individual sequences via variable-rate coding”. In: *IEEE Transactions on Information Theory* 24.5 (1978), pp. 530–536. DOI: `10.1109/TIT.1978.1055934`.